DS 1

Option informatique, deuxième année

Julien Reichert

Exercice 1 : Écrire en Caml une fonction qui prend en argument une liste `l` (dont on note n la taille) et qui retourne la liste des sous-listes de `l` dont les éléments sont dans le même ordre (il y en a 2^n).

Exercice 2 : Écrire en Caml une fonction de signature `successeur : 'a ABR -> 'a -> 'a` telle que `successeur a elt` renvoie l'élément de l'arbre `a` directement après `elt`. On doit recevoir une exception ou un message d'erreur si `elt` n'est pas dans `a` et une autre exception ou message d'erreur si `elt` est le plus grand élément de `a`.

L'implémentation obligatoire est le type suivant :

```
type 'a ABR = Vide | Noeud of 'a ABR * 'a * 'a ABR
```

Exercice 3 : Écrire en Caml une fonction qui prend en entrée un arbre binaire (déclarer un type au choix ou signaler que le précédent est réutilisé) et qui détermine s'il s'agit d'un ABR. Déterminer la complexité dans le pire des cas de la fonction, il faut que la réponse soit au maximum quadratique en la taille de l'arbre.

Exercice 4 : Écrire en Caml une fonction qui prend en entrée un arbre et qui détermine s'il s'agit d'un tas-min.

Pour l'exercice 4, on impose le type suivant :

```
type 'a arbre = Noeud of 'a * 'a arbre list;;
```

Il n'y a cette fois-ci pas d'arbre vide possible, mais cela simplifie le reste, et une feuille sera donc par exemple un nœud pour lequel la liste associée est vide.

Exercice 5 : On considère une structure d'arbre binaire spéciale, dans laquelle chaque nœud porte une information supplémentaire, la hauteur du sous-arbre enraciné en ce nœud.[1] Écrire en Caml une fonction qui prend en entrée un arbre de cette forme et une valeur et qui retourne l'arbre obtenu en insérant la valeur dans l'arbre de départ, sachant que le sous-arbre de hauteur la plus faible est choisi à chaque étape. Obtient-on alors nécessairement un arbre de hauteur minimale ?

On utilisera le type suivant :

```
type 'a arbre_hauteur = Vide | Noeud of 'a arbre_hauteur * ('a * int) * 'a arbre_hauteur;;
```

1. Une telle structure peut être utilisée à des fins d'équilibrage.

Exercice 6 : Avec les types ci-dessous définissant de trois façons différentes un graphe, écrire une fonction convertissant une représentation de type `graphe3` d'un graphe en la représentation de type `graphe2` du même graphe (en créant des noms de sommets au choix mais cohérents) et une fonction convertissant une représentation de type `graphe2` d'un graphe en la représentation de type `graphe1` du même graphe (en utilisant les mêmes noms de sommets).

```
type graphe1 = { mutable sommets : string list;
mutable arcs : (string * string) list; };;
type graphe2 = (string * (string list)) list;;
type graphe3 = bool array array;;
```

Exercice 7 : Déterminer le plus court chemin du sommet `A` au sommet `G` dans le graphe pondéré (non orienté) dessiné en figure 1. L'algorithme utilisé sera au choix parmi Dijkstra, Floyd-Warshall et Bellman-Ford, et il faudra détailler les étapes de calcul.

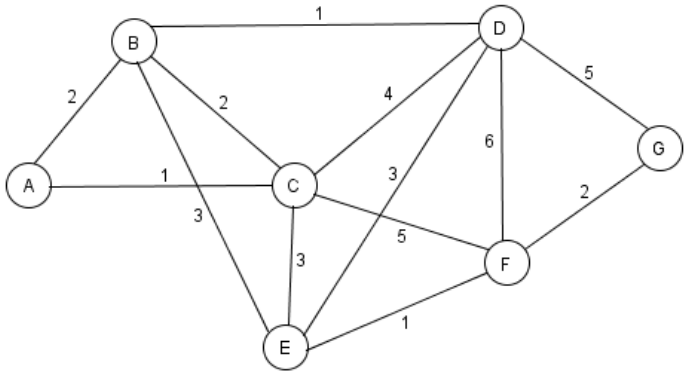

FIGURE 1 – Graphe pondéré

Pour éviter de s'ennuyer, deux exercices supplémentaires !

Exercice 8 : Écrire un programme en Caml prenant en entrée un graphe orienté (représentation au choix) et retournant un chemin eulérien donné en tant que liste de sommets, s'il en existe un. S'il n'en existe pas, on retournera au choix une liste vide ou une erreur.

Exercice 9 : Écrire un programme en Caml prenant en entrée un graphe non orienté (représentation au choix) et retournant un chemin hamiltonien donné en tant que liste de sommets, s'il en existe un. S'il n'en existe pas, on retournera au choix une liste vide ou une erreur. Si la complexité semble énorme, il ne faut pas s'inquiéter...